Stork User Manual

version 1.2.1

1 Building and installing Stork

Overview (quick start)

If you prefer not to read this entire section and just want to install a copy of Stork as soon as possible, simply paste these commands into a shell, replacing *<install-path>* with the path to the directory in which you want Stork to be installed.

To install Stork from source:

Goto www.storkproject.org/downloads and download the appropriate files

```
$ tar xvf stork-1.2.1.tar.gz
$ cd stork-1.2.1/
$ ./configure --prefix=<install-path>
$ make install
$ export STORK_CONFIG=<install-path>/etc/stork_config
$ export PATH=$PATH:<install-path>/bin:<install-path>/sbin
```

Or to install the binary package: Goto www.storkproject.org/downloads and download the appropriate files

```
$ tar xvf stork_binary-1.2.1.tar.gz
$ cd stork_binary-1.2.1/
$ ./stork-install.ksh <install-path>
$ export STORK_CONFIG=<install-path>/etc/stork_config
$ export PATH=$PATH:<install-path>/bin:<install-path>/sbin
```

Unless any errors occurred, Stork should now be installed on your system. You may start the Stork server by running `stork_server`. If you did not specify

an installation path, Stork will by default be installed to `/usr/bin/local`.

1.1 Download and extract

Stork may be downloaded from the Stork project homepage by following the Downloads link. You will have the option to download either the source version or a pre-compiled binary package.

If you are a user of a unix-like system, it is recommended that you download the source code, as this will allow you to better customize your installation. If you are running on a system on which Stork is not known to compile, or you find you are not able to compile from source for other reasons, you may want to see if a binary package is available for your system.

After downloading a package which suits your needs, simply extract the package with whatever tools you have at your disposal for decompressing and untarring gzipped tar archives. Most users would simply use the tar command. For example, to extract the Stork 1.2.1 source:

```
$ tar xvf stork-1.2.1.tar.gz
```

This will create a directory in your current directory containing the source code. Afterwards, simply `cd` into the directory.

If you chose to download the binary package, you may now skip to the Installation section. Otherwise, continue onward.

1.2 Externals

If you plan on building contributed (third-party) transfer modules for Stork, you will need to download the externals package from the Stork website, and place its contents into the `./externals/bundles/` directory. If you have no such plans, you may skip this step completely.

The externals package, at this point, contains the following bundles: globus, irods, openssl, and srb. The following table shows the protocols with available transfer modules and the bundles required to compile them:

Protocol	Bundles
FTP	globus, openssl
GSIFTP	globus, openssl
HTTP	globus, openssl
iRODS	irods, srb
SRB	srb

Please note that Stork uses Globus components for file transfers using the *GSIFTP* protocol (GridFTP). Therefore, you need to have the Globus Toolkit installed and configured on your system with user/host certificates in order to use this transfer module.

1.3 Configuring

After downloading and extracting the source, you must configure the source tree with the `configure` script therein. Note, however, that if you are trying to build a development version of Stork (odd minor version numbers), then you will need to have `autoconf` and `autoheader` (at least version 2.59) installed in order to first generate the `configure` script. This is unnecessary if you are trying to build a release, as releases already contain a `configure` script.

To build the Stork client commands, the server, and the core (supported) transfer modules, and install to the default `/usr/local/stork` directory, simply run `./configure` with no options. However, if you want to build contributed (third-party) transfer modules, want to build Stork with other features, or need to specify a special compiler or library to use, you must use additional options with `./configure`. (A complete list of options may be seen by running

`./configure --help`.)

You will probably want to specify a location other than the default location (`/usr/local/stork`) to install Stork. If you are not a superuser, this is a necessity. This can be done by specifying `--prefix=`*`path`* with `./configure`, where *`path`* is the directory in which you want to install Stork.

You might also want to build contributed transfer modules along with Stork. This can be done by specifying `--with-`*`bundle`* with `./configure`, where *`bundle`* is the name of the external package necessary to build the transfer module in question. If you specify *all* of the necessary bundles for a given transfer module (as specified in the table in the Externals section) with `--with-`*`bundle`*, and the bundles can all be found in the `./externals/bundles/` directory, then the transfer module in question will be built. More information about this can be found in the Externals section above and by running `./configure --help`.

`configure` will take a few minutes to check that everything necessary for building Stork is available on your system. After it is done running, you are ready to move on to the next step.

1.4 Building

Building everything is simple: just run `make`.

Depending on the speed of your system and the number of externals you require, building may take anywhere from 3 to 20 minutes. Just be patient. Note that while external modules are building, output will stop for a significant amount of time. Keep in mind that this is normal, and don't worry that your screen may have frozen. If you are interested in watching the output from those external builds for some reason, you may do so by running (in another terminal, of course) `tail -f` with the path to the build log shown before the module begins building given as an argument.

1.5 Installation

If you have been building from source up to this point, all you have to do now is run `make install`. This will install everything to the location specified with `--prefix` when you ran `./configure`, or to `/usr/local/stork` if you did not specify anything. After it has installed (it shouldn't take long), you may go to the next step.

If you are installing from a binary package, you simply have to run the included installation script with the desired installation path *<install-path>* as an argument: `./stork-install.ksh` *<install-path>*

This will extract the included binaries and place them in the location specified by *<install-path>*.

1.6 Post-installation

Now that you have installed Stork, you must configure your environment in order to use Stork with minimal effort. Regardless of whether you built Stork from source or downloaded the binaries, after the script you ran in the previous step completes, you will be shown a message asking you to set your `PATH` and `STORK_CONFIG` environment variables by running commands similar to these:

```
$ export STORK_CONFIG=/path/to/stork/etc/stork_config
$ export PATH=$PATH:/path/to/stork/bin:/path/to/stork/sbin
```

You should do as the script says! In order to make sure that your environment is configured properly every time you start a new shell, you should also add these commands to a start-up script (e.g., the `.bashrc` file in your home directory). This will prevent you from needing to run these commands every time you want to use Stork.

If you have reached this point without a problem, you now have a working Stork installation. :) You may test this by starting a Stork server with the

command `stork_server`. If you would like to edit the `stork_config` file and configure your Stork installation further, it may be found in the `etc` directory inside the newly installed Stork directory.

2 Running Stork

If a user called `stork` exists, the server will switch to the `stork` user for security purposes.

The command below starts the Stork server connected to port `<port>` The stork logs are named with the prefix Stork, such as StorkLog, Stork.history, etc.

```
stork_server -p <port> -Serverlog <stork/log/directory/Stork>
```

The Stork server generates a log file which is used for logging its activities. Below is a sample of the log file:

```
cat local/log/StorkLog
0/23 16:55:24 ******************************************************
10/23 16:55:24 ** stork_server (STORK) STARTING UP
10/23 16:55:24 ** /tmp/sbin/stork_server
10/23 16:55:24 ** $CondorVersion: 6.9.4 Sep 11 2008 $
10/23 16:55:24 ** $CondorPlatform: I386-LINUX_RHEL3 $
10/23 16:55:24 ** PID = 13697
10/23 16:55:24 ** Log last touched time unavailable (No such file or directory)
10/23 16:55:24 ******************************************************
10/23 16:55:24 Using config source: /tmp/etc/stork_config
10/23 16:55:24 DaemonCore: Command Socket at <208.100.92.21:47661>
10/23 16:58:21 ******************************************************
10/23 16:58:21 ** stork_server (STORK) STARTING UP
10/23 16:58:21 ** /tmp/sbin/stork_server
10/23 16:58:21 ** $Version: 6.9.4 Sep 11 2008 $
10/23 16:58:21 ** $Platform: I386-LINUX_RHEL3 $
10/23 16:58:21 ** PID = 14336
10/23 16:58:21 ** Log last touched 10/23 16:55:24
10/23 16:58:21 ******************************************************
10/23 16:58:21 Using config source: /tmp/etc/stork_config
10/23 16:58:21 DaemonCore: Command Socket at <208.100.92.21:47712>
10/23 16:58:21 ============================================================
10/23 16:58:21               STORK CONFIGURATION:
10/23 16:58:21 ============================================================
10/23 16:58:21 DaP log file     : storkserver.log
10/23 16:58:21 Userlog file     : (null)
10/23 16:58:21 XML log file     : (null)
10/23 16:58:21 Client Agent host: (null)
10/23 16:58:21 ============================================================
10/23 16:58:21 STORK_TEST_MODE = 10 (0: do testing only - 10: run as server)
10/23 16:58:21 STORK_MAX_NUM_JOBS = 1

10/23 16:58:21 STORK_MAX_RETRY = 1
10/23 16:58:21 STORK_MAXDELAY_INMINUTES = 0
10/23 16:58:21 STORK_AGGR_LEVEL = 0 - 0: no aggregation
```

```
- 1: according to dest url; 2: according to src url - 3: if hostnames match
10/23 16:58:21 STORK_MAX_COUNT = 1 -
10/23 16:58:21 STORK_RECURSIVE_COPY = FALSE
10/23 16:58:21 STORK_VERIFY_CHECKSUM = FALSE
10/23 16:58:21 STORK_VERIFY_FILESIZE = FALSE
10/23 16:58:21 STORK_NETWORK_CHECK = FALSE
10/23 16:58:21 STORK_TRANSFER_CHECKPOINT = FALSE
10/23 16:58:21 STORK_SYNC_ONLY = FALSE
10/23 16:58:21 STORK_TMP_CRED_DIR = /tmp
10/23 16:58:21 STORK_MODULE_DIR = /tmp/libexec
10/23 16:58:21 modules will execute in LOG directory /tmp/local/log
10/23 16:58:21 Getting monitoring info for pid 14336
.....................
.....................
```

3 Stork Components

stork_server:

The Stork server is the main component of the Stork scheduler. The Stork server runs as a persistent daemon process and performs all management, scheduling, execution, and monitoring of data placement activities.

The Stork server accepts the following parameters (defined by `STORK_ARGS`)

```
$ sbin/stork_server --help
============================================================
USAGE: stork_server
         [ -t           ] // output to stdin
         [ -p           ] // port on which to run Stork Server
         [ -help        ] // stork help screen
         [ -Config      ] // stork config file
         [ -Serverlog   ] // stork server log in ClassAds
         [ -Xmllog      ] // stork server log in XML format
         [ -Userlog     ] // stork userlog in XMLformat
         [ -Clientagent ] // host where client agent is running
============================================================
```

stork_submit:

`stork_submit` is a client-side tool used to submit stork jobs to the Stork server.

```
$ bin/stork_submit
usage: stork_submit  [option]... [stork_server] submit_file
stork_server                    specify explicit stork server (deprecated)
submit_file                     stork submit file
       -lognotes "notes"       add lognote to submit file before processing
```

```
       -stdin                  read submission from stdin instead of a file
       -help                   print this help information
       -version                print version information
       -debug                  print debugging information to console
       -name stork_server      stork server
```

stork_status:

`stork_status` is a client side tool used to query regarding the status of jobs submitted to the Stork server.

The `dap_id` is used by the `stork_status` commmand to query the Stork server. The `dap_id` is generated and assigned to a job when it is submitted to Stork using the `stork_submit` command.

The `stork_status` command accepts the following parameters, where `host_name` is optional. The `host_name` is used to specify a Stork server on a remote host.

```
./stork_status -h
usage: stork_status  [option]... [stork_server] job_id
stork_server                    specify explicit stork server (deprecated)
job_id                          stork job id
       -help                   print this help information
       -version                print version information
       -debug                  print debugging information to console
       -name stork_server      stork server
```

stork_rm:

`stork_rm` is a client side tool used to delete any jobs that are currently queued with the Stork server.

```
./stork_rm -h
usage: stork_rm  [option]... [stork_server] job_id
stork_server                     specify explicit stork server (deprecated)
job_id                           stork job id
       -help                   print this help information
       -version                print version information
       -debug                  print debugging information to console
       -name stork_server      stork server
```

stork_q:

`stork_q` is a client side tool used to retrieve a listing of jobs that are currently queued with the Stork server.

```
./stork_q -h
usage: stork_q  [option]... [stork_server]
stork_server                    stork server (deprecated)
       -help                   print this help information
       -version                print version information
       -debug                  print debugging information to console
       -name stork_server      stork server
```

Sample output from the `stork_q` command:

```
[
    dest_url = "file:/home/user1/stork/data10M_48";
    src_url = "file:/home/user1/stork/data10M";
    remote_user = "user1@dsl-turtle06.cct.lsu.edu";
    status = "request_rescheduled";
    dap_id = 264;
    use_protocol = 0;
    stork_server = "qb1.loni.org";
    dap_type = "transfer";
    error_code = "port not accessible";
    num_attempts = 1;
    owner = "user1";
    cluster_id = 264;
    timestamp = absTime("2008-05-28T14:52:22-0500");
    generic_event = "Rescheduling."
]
```

Stork server runs as a persistent daemon process. It constantly listens to requests from the clients. The clients send their requests to the Stork server over the network using `stork_submit` command line tool in form of a ClassAd (Classified Advertisement).

Since Stork is designed to work in a heterogeneous computing environment, one of its goals is to support as many storage systems and file transfer protocols as possible.

Another important characteristic of Stork is reliability. It makes sure that

the requested transfers are completed successfully even in case of server or network failures.

Stork source and destination URLs have a naming convention. All URLs ending with a slash (/) are assumed to be directories and the rest are assumed to be files.

Job Submission

Currently the following protocols and storage systems are supported by Stork:

- file
- FTP
- GridFTP
- HTTP
- iRODS
- PetaShare
- SRB

The protocol to be used is determined by the Stork server according to the URL signatures of the files to be transfered.

URLs supported: The format of the URL for various supported protocols is as below:

- file URL - file:/path/to/file
- FTP URL - ftp://user:password@host:port/path/to/file
- HTTP URL - http://user:password@host:port/path/to/file
- GridFTP URL - gsiftp://user:password@host:port/path/to/file
- SRB URL - srb://user[.mdasDomain[.zone]]:password@host:port/path/to/file
- iRODS URL - irods://user.zone:password@host:port/path/to/file
- PetaShare - petashare://path/to/file

Assuming that a service (SRB, iRODS, GridFTP, etc) is running at host on port. Please note that in the URLs shown above the parameters denote:

- user - the username
- password - password corresponding to the username
- host - the host on which the service is running
- port - the port the service listens on
- /path/to/file - path to the location of the file you would like to transfer
- /path/to/directory/ - path to the directory you would like to perform transfers

Sample Stork Job Requests

i) file to file transfer

```
[
  dap_type = "transfer";
  src_url = "file:/path/to/file";
  dest_url = "irods://user.zone:password@host:port/path/to/file";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/file";
  dest_url = "file:/path/to/file";
]
```

ii) file selection using wild cards

```
[
  dap_type = "transfer";
  src_url = "file:/path/to/file*";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "file:/path/to/*file";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]
```

```
[
  dap_type = "transfer";
  src_url = "file:/path/to/fi*le";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "file:/path/to/*";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/file*";
  dest_url = "file:/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/*file";
  dest_url = "file:/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/fi*le";
  dest_url = "file:/path/to/directory/";
]

[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/*";
  dest_url = "file:/path/to/directory/";
]
```

iii) recursive transfer from local directory to SRB collection

```
[
  dap_type = "transfer";
  src_url = "file:/path/to/directory/";
  dest_url = "irods://user.zone:password@host:port/path/to/directory/";
]
```

```
[
  dap_type = "transfer";
  src_url = "irods://user.zone:password@host:port/path/to/directory/";
  dest_url = "file:/path/to/directory/";
]
```

Any of the supported protocols may be invoked by simply replacing the URLs shown above by those of the protocols required. The new URLs should however conform to their URL format as described in the supported URL format section above.

4 New Features

Here is an example submit file including extended features.

```
etc/submitArgument.sample
[
       dap_type = "transfer";
       src_url  = "gsiftp://$src.loni.org/home/balman/tests/$srcfile";
       dest_url = "gsiftp://$dest.loni.org/home/balman/tests/dest-$destfile";
       output   =  "out";
       err      =  "err";
       arguments = "-p 10";
       set_permission           = "066";
       sync_only                = true;
       checkpoint_transfer      = true;
       network_check            = true;
       verify_filesize          = true;
       recursive_copy           = true;
]
```

File size verification support

Currently all the transfer modules supported by Stork support file size verification. File size verification can either be turned on or off by specifying the corresponding option in the Stork configuration file.

When switched on, Stork determines the file sizes of the files at the source and the files at the destination and compares them. If the file sizes differ, an

error message is logged to the Stork log file.

Checksum verification support

Currently all the transfer modules supported by Stork support checksum verification. Checksum verification can either be turned on or off by specifying the corresponding option in the Stork configuration file.

When switched on, Stork computes the checksums of the files at the source and the files at the destination and compares them. If the checksums differ, an error message is logged to the Stork log file.

Recursive transfers

Currently all the transfer modules supported by Stork support recursive directory transfers. Recursive directory transfers are specified in the URL by ending the URLs with a '/' to represent a directory.

Wild card support

Currently all the transfer modules supported by Stork support transferring files with a wild cards such as *.txt, stork*, *stork or st*rk.

Checkpointing File Transfers

Currently the Petashare and GridFTP transfer modules supported by Stork support checkpointing of transfers and provide the capability of resuming transfers in the event of an error.

These Stork modules checkpoint the transfers during various stages and thus enable Stork to resume the transfer at the last checkpoint in the event of a network outage or crash.

5 Running Stork in different machines

Stork provides the flexibility of running stork- server in one location and stork - client in another location. This feature will be useful for many situations where a server can't run for extended period of time due to server access time limitations

Consider the following example. In this case, we are going to run `stork_server` in $dsl - condor$ machine and stork client in another machine called `dsl-stork`. stork server is located in your sbin directory of the stork installation and clients are located in bin directory.

Step 1: Start stork server in $dsl - condor$ by running the following command

`stork_server -p port number` (Port number is the one you are going to open for the connection)

For example, `{ stork_server -p 9621}`

Step 2: In Client machine, you need to edit stork-config file. Declare `{Stork_host = destination:port number}`

Step 3: Submit your jobs to the stork server from the client machine

`{stork_submit dap file -name storkserveraddress : port number}`

For example, `{stork_submit test.dap -name dsl-condor.lsu.edu:9621}` Stork then returns the associated job id, which is used by other Stork job control tools.

Step 4: You can do the regular stork client operation such as `{stork_status, stork_q, stork_rm }` in client's mchine

Step 5: You can look for server log entries in server side.

Stork fully supports GridFTP transfers, with the gsiftp:// protocol. To use GridFTP with Stork, users must have a valid proxy. Specify the path or say default to the created proxy using the x509proxy keyword in the Stork submit

file. Placing the special value **x509proxy = " default";** or **x509proxy = " /tmp/x509up-uxxxx "**

Also, for GSIFTP transfers, you need to have client grid proxy initiated and server grid proxy initiated. Stork will use the users credentials to authenticate to the GridFTP server.

For example,

```
[
      dap_type = "transfer";
      src_url  = "gsiftp://$src.loni.org/home/sivahpc/test/$srcfile";
      dest_url = "gsiftp://$dest.dsl-stork.org/home/sivahpc/test/dest-$destfile";
      x509proxy = "default";
      ]
```

6 Questions

Direct your questions to stork-devel@mail.cct.lsu.edu, if you have any problems related to stork

7 Appendix

Platforms supported by Stork -1.2.1

The following are the list of transfer modules that are supported by Stork in each of the platforms listed below.

Platforms	Core Stork	GridFtp	SRB	iRODS	Petashare
ia64_rhas_3	Yes	Yes	Yes	No	No
ia64_rhas_4	Yes	Yes	Yes	No	No
ia64_sles_9	Yes	Yes	Yes	No	No
x86_64_fedora_8	Yes	Yes	Yes	Yes	Yes
x86_64_rhap_5	Yes	Yes	Yes	Yes	Yes
x86_64_rhap_5.2	Yes	Yes	Yes	Yes	Yes
x86_64_rhap_5.3	Yes	Yes	Yes	Yes	Yes
x8664rhap5.3upd	Yes	Yes	Yes	Yes	Yes
x86_64_rhas_3	Yes	Yes	Yes	Yes	Yes
x86_64_rhas_4	Yes	Yes	Yes	Yes	Yes
x86_64_sles_9	Yes	Yes	Yes	Yes	Yes
x86_64_macos_1(	Yes	Yes	No	No	No
x86_64_deb_4.0	Yes	Yes	Yes	Yes	Yes
x86_deb_4.0	Yes	Yes	Yes	No	No
x86_rhap_5.0	Yes	Yes	Yes	No	No
x86_rhas_3.0	Yes	Yes	Yes	No	No
x86_rhas_4.0	Yes	Yes	Yes	No	No
x86_sles_9	Yes	Yes	Yes	No	No
x86_ubuntu_5.10	Yes	Yes	Yes	No	No
x86_macos_10.4	Yes	No	No	No	No